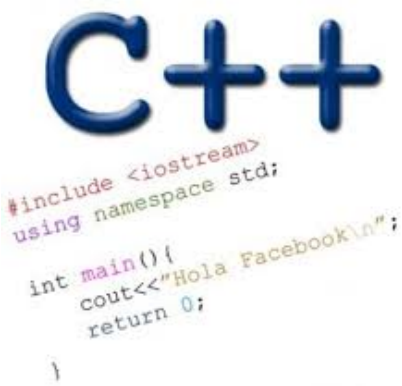
FILE IO
TEST DRIVEN DEVELOPMENT
MAKEFILES

Problem Solving with Computers-I



I/O in programs

Different ways of reading data into programs

- Standard input (stdin) using cin
- Command line arguments (int main(int argc, char* argv[])
- Read from file

Ways to output data

- Standard output: cout
- Standard error: cerr
- Write to file

Writing to files

```
#include <fstream>
ofstream ofs;  // Create a ifstream object
ofs.open("animals.txt"); //Open a file to write to
ofs<<"Duck\n"<<"Cat\n"<<"Cow\n";
```

Reading from files

- Open a file
- If open fails, exit
- In a loop
  - Read a line
  - If you reach the end of file, break
  - Else process the line that was read
- Close the file

Reading from files

```
#include <fstream>
ifstream ifs;  // Create a ifstream object
ifs.open("numbers.txt"); //Open a file to read
if(!ifs){
      // open failed
}
getline(ifs, line); // read a line from the file into a
                    // string line.
                    // If you attempt to read past the end
                     // of file, ifs change to false

// If the file was empty, ifs will be false at this point
ifs.close()
```

FILE IO: Which of the following is correct?

A.

```
while(1){
        getline(ifs, line);
        if (!ifs)
            break;
        cout<<line<<endl;
}
```

B.

```
while(ifs){
        getline(ifs, line);
        cout<<line<<endl;
}
```

C. Both A and B are correct

D. Neither is correct

Lab02 practice & TDD

Write a function that RETURNS a string representing an isosceles triangle with a given width.

We will use this example to introduce test driven development

```
s = drawTriangle(5);
cout<<s;

   *
  ***
 *****
*******
```

Make and makefiles

- The unix make program automates the compilation process as specified in a Makefile
- Specifies how the different pieces of a program in different files fit together to make a complete program
- In the makefile you provide a recipe for compilation
- When you run make it will use that recipe to compile the program

```
$ make
g++ testShapes.o shapes.o tdd.o -o testShapes
```

Specifying a recipe in the makefile

- **Comments** start with a #
- **Definitions** typically are a variable in all caps followed by an equals sign and a string, such as:

```
CXX=g++
CXXFLAGS=-Wall

BINARIES=proj1
```

```
# testShapes is the target - it is what we want to produce
# To produce the executable testShapes we need all the .o files
# Everything to the right of ":" is a dependency for testShapes

testShapes: testShapes.o shapes.o tdd.o
	#The recipe for producing the target (testshapes) is below
	g++ testShapes.o shapes.o tdd.o -o testShapes
```

Demo

- Basics of code compilation in C++ (review)
- Makefiles (used to automate compilation of medium to large projects) consisting of many files
- We will start by using a makefile to compile just a single program
- Extend to the case where your program is split between multiple files
- Understand what each of the following are and how they are used in program compilation
  - Header file (.h)
  - Source file (.cpp)
  - Object file (.o)
  - Executable
  - Makefile
  - Compile-time errors
  - Link-time errors

Next time

- Data Representation